Quantifying Scheduling Challenges for Exascale System Software

Oscar H. Mondragon
Department of Computer Science
University of New Mexico
Albuquerque, NM 87131
omondrag@cs.unm.edu

Patrick G. Bridges
Department of Computer Science
University of New Mexico
Albuquerque, NM 87131
bridges@cs.unm.edu

Terry Jones
Computer Science and Mathematics Division
Oak Ridge National Laboratory
TN 37831, USA
trjones@ornl.gov

ABSTRACT

The move towards high-performance computing (HPC) applications comprised of coupled codes and the need to dramatically reduce data movement is leading to a reexamination of time-sharing vs. space-sharing in HPC systems. In this paper, we discuss and begin to quantify the performance impact of a move away from strict space-sharing of nodes for HPC applications. Specifically, we examine the potential performance cost of time-sharing nodes between application components, we determine whether a simple coordinated scheduling mechanism can address these problems, and we research how suitable simple constraint-based optimization techniques are for solving scheduling challenges in this regime. Our results demonstrate that current general-purpose HPC system software scheduling and resource allocation systems are subject to significant performance deficiencies which we quantify for six representative applications. Based on these results, we discuss areas in which additional research is needed to meet the scheduling challenges of next-generation HPC systems.

Categories and Subject Descriptors

D.4.7 [**Operating Sytems**]: Organization and Design; C.5.1 [**Computer System Implementation**]: Super (very large) computers; C.1.2 [**Multiprocessors**]: Parallel Processors

Keywords

Scheduling, time-sharing, performance

1. INTRODUCTION

System software stacks for next-generation high-performance computing (HPC) systems face a range of scheduling and resource allocation problems due to changes in applications and hardware. For example, large-scale scientific applications are beginning to perform simulation and analysis concurrently instead of sequentially. Also, hardware memory and network bandwidth restrictions and potential system power caps motivate the need to minimize data movement and power down unneeded hardware components whenever possible.

Together, these constraints are leading to a reexamination of the strict spatial allocation of HPC nodes to single application codes. New usage models attempt to co-locate related functional components onto single node in order to avoid data movement and meet power budgets [6]. Recent research has also demonstrated the potential of time-sharing processors between related application and analytics codes, provided they are appropriately scheduled [30].

As an example of this trend, consider the *exploratory analytics* application [6] shown in Figure 1. Each portion of the coupled-application in this example runs in its own *enclave*, a set of system resources allocated to an application [2]. In this example, Enclave 1 is a simulation enclave with components running on nodes 1 and 2, and Enclave 2 contains an analysis code co-located with Enclave 1 in node 1. Each enclave has its own *Enclave OS (EOS)* that provides a runtime customized to its needs. The underlying node operating system allocates resources to each enclave running on the node, potentially based on information provided by a higher-level system-wide resource allocation system. The higher-level component is similar to the provisioner used in modern cloud computing systems to partition a node between virtual machines. Efficient operation depends on all three components (system, enclave, and node-level operating systems) managing complex hardware and application features including the impacts of limited memory bandwidth, frequency and voltage scaling, and system power caps.

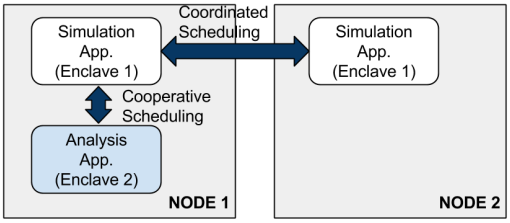

Figure 1: Example. Exploratory analytics application

Neither current HPC nor Cloud scheduling systems ad-

ROSS '15, June 16, 2015, Portland, Oregon, USA

ACM 978-1-4503-3606-2/15/06 ...$15.00.
http://dx.doi.org/10.1145/2768405.2768413

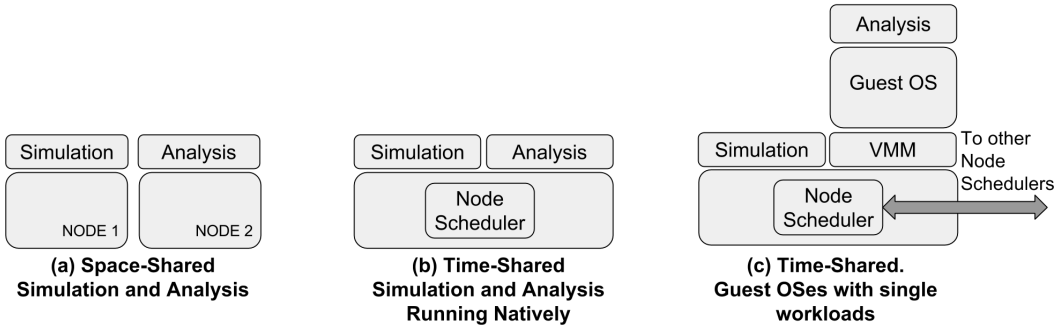

Figure 2: Different resource allocation approaches

dress many of the scheduling and resource allocation problems presented by this example. Existing HPC resource allocation is generally *space-shared*, with each enclave mapped to its own node. In addition, current HPC and Cloud systems generally support limited scheduling policies even though NUMA constraints and power caps are increasingly important. Finally, while some Cloud software systems share nodes using hardware virtualization techniques, these systems fully isolate virtual nodes from one another, and are not designed for cooperative scheduling between virtual machines or cross-node gang scheduling techniques.

In this paper, we provide the first evaluation of the potential performance impact of each of these scheduling challenges by assessing them on six representative HPC applications. Specifically, we characterize the new scheduling challenges faced by exascale applications on time-shared systems and provide a quantitative evaluation of the impact of those challenges through experiments using a set of applications. Finally, we discuss research challenges in scheduling that must be addressed for emerging exascale architectures.

The remainder of this paper is organized as follows. In Section 2, we provide additional background on the new scheduling challenges faced by exascale systems. In Section 3, we begin to quantify the potential impact of these challenges in three areas: the cost of uncoordinated time-sharing on HPC application performance, the viability of node-level coordinated scheduling to address these costs, and the ability of recently-proposed quadratic programmings to solve the resource allocation problem. Based on these results, Section 4 analyzes and discusses potential scheduling research directions in exascale operating systems. Section 5 then discusses related work in this area, and Section 6 concludes.

2. NEW SCHEDULING CHALLENGES IN EXASCALE SYSTEMS

In current HPC systems, resource allocation is *space-shared*. For example, applications performing in-situ analytics frequently use a subset of the entire compute node set for analytics. This leads to expensive data movement between simulation nodes and analytics nodes, as shown in Figure 2(a). Even when the analytics portion is co-located on nodes with the simulation portion, as shown in Figure 2(b), the OS services required to support complex analytics may result in OS noise that slows down application simulation [8]. Finally, pure spatial partitioning of a node can result in excess data movement between processors and NUMA memory banks in the node.

These features have led to recent HPC system software designs that propose using virtualization to separate the simulation OS from the analytics OS on the same node [6, 14]. Figure 2(c) shows this example. Similarly, researchers have recently demonstrated that it is possible to cooperatively schedule analytics and simulation on the same nodes with minimal impact of the analytics on simulation [30].

The combination of these two techniques portend a major change in scheduling and resource allocation for HPC systems and present major scheduling challenges. In the remainder of this section, we discuss the implications of these and related trends to scheduling in HPC system software.

2.1 Efficient Node-level Resource Allocation

Co-locating and cooperatively scheduling multiple enclaves within a single node complicates the already difficult problem of node-level resource allocation. In particular, the scheduler must determine which enclave processes to place on the same processor while simultaneously taking into account NUMA memory placement. The appropriate scheduling criteria is also complex, involving energy and performance tradeoffs, as well as power constraints.

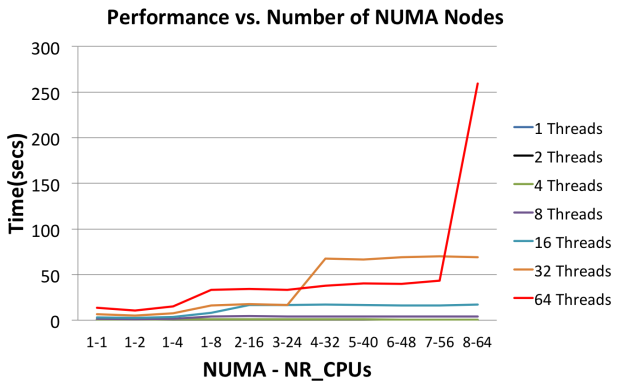

Figure 3: HPCC random access benchmark performance for different numbers of NUMA domains. 1-2 stands for one NUMA domain and two cores.

Various numerical optimization approaches have been recently proposed as potential solutions. For example, re-

searchers have proposed using techniques from bin-packing heuristics [28], genetic algorithms [22], and convex optimization [4] to address this problem. Successfully solving these problems, however, has proved troublesome and much of the work described above has focused only on single applications, not more complex coupled codes.

To illustrate the importance of resource allocation in even simple examples, we ran the HPCC Random Access benchmark [19] on the Susitna-PRobE machines [10]. For this simple demonstration, we co-locate threads of a single application. Figure 3 shows the results obtained from running the benchmark with different numbers of threads and NUMA domains. In this figure, the x-axis shows the number of NUMA domains and the total number of CPUs used to run the benchmark and y-axis shows the time to solution. These results show that the performance of even this simple benchmark is highly sensitive to scheduling decisions. For example, from the 64-threads test, we found that performance is improved if we restrict the threads to a single NUMA domain. For this case, performance is further improved if we use just one physical core instead of eight. More generally, this simple example shows the need of scheduling frameworks that provide proper interfaces and mechanisms to allocate resources to applications based on data locality and resource contention behavior.

2.2 Efficient Intra-node and Inter-node Synchronization/Coordination Mechanisms

Next-generation system software must perform in increasingly complex environments where synchronization/coordination mechanisms may introduce new requirements between node-level schedulers, as illustrated in Figure 2(c). For example, codes that perform either intra-node or inter-node communication may require gang-scheduling strategies to minimize performance slowdowns associated with blocking communication. Blocking collectives remain a stalwart of HPC application design, and the growing scale of HPC systems potentially dramatically increases the need for techniques to mitigate OS noise. This is particularly true for dynamic applications that magnify OS noise when they become load-imbalanced. To make matters worse, the complexity and scale of emerging computer architectures are leading to system resilience techniques that yield a large amount of additional interference.

It is not clear if previous strategies [13, 16, 26, 27] for scheduling synchronization-sensitive applications are sufficient for solving this problem. *Vsched* [16], for example, uses an Earliest Deadline First (EDF) scheduler-based approach [18] to provide gang scheduling support; however, it does not consider co-location with cooperative workloads, nor the dissimilarity of *utilization factors* across the processors in the gang.

2.3 Co-location of Cooperative Enclaves

In addition to coordinated scheduling across nodes and between cores within a node, efficient time-sharing of processor resources will require careful scheduling of cooperating enclaves. Recent work on co-locating analytics and simulation in the Goldrush system [30], for example, has shown that co-located analytics that are co-scheduled with simulation can dramatically reduce simulation performance without special measures. The Goldrush work addresses this at the application level by explicitly yielding the processor between processes running in the same operating system; introducing multiple enclave OSes could significantly complicate such efforts, and more general OS-level scheduling mechanisms to support such systems (e.g., more general and usable than user-driven `SIGSUSPEND`) are also very desirable.

3. IMPACT OF THE NEW SCHEDULING CHALLENGES ON HPC APPLICATIONS PERFORMANCE

We conducted a variety of experiments to understand the scope of the challenges on HPC applications performance, as well as to gain insight into the potential power of some previously-proposed solutions. First, we examined a simple numerical scheduling technique for node-level resource allocation. We then examined the potential for traditional gang scheduling techniques inside virtualized nodes, and how such techniques translate to systems with cooperative scheduling needs derived from co-located workloads. Each of these experiments were conducted in the context of the Palacios virtual machine monitor [14] using Linux as the host OS.

3.1 Numerical Optimization of Node-level Resource Allocation

Constrained optimization mechanisms have recently been proposed as a potential technique for resource allocation in HPC systems. Such techniques are attractive because they can handle a range of optimization criteria and constraints. When the optimization problem is convex, a wide range of relatively inexpensive numerical techniques may be used to solve these problems. When the problem is non-convex, approximation techniques can be used.

To provide an initial evaluation of the viability of such techniques, we explored the use of constrained quadratic programming (QP) to address processor allocation challenges in HPC systems. The goal of Quadratic Programming is to minimize a quadratic *objective function* with linear constraints [5]. We sought to use this technique to efficiently map Palacios virtual cores running an HPC application to the underlying NUMA domains, sockets, and cores.

3.1.1 Problem Formulation Process

We considered a number of possible formulations for the underlying quadratic programs; convex quadratic programs generally failed to encompass important aspects of the problem. Non-convex quadratic programs, in contrast, were able to encompass all aspects of the problem, but were more computationally expensive to solve (non-convex quadratic programs are potentially NP-complete to solve precisely).

In a first approach, we attempted to state our problem as a convex quadratic program, as proposed in [4], which permits fast and scalable solutions. Our *optimization variable* was a vector that contains the percentage of a physical core allocated to each virtual core. However, given the variety of factors that must be considered in the objective function, convexity can not be guaranteed.

A second approach relied on framing the problem as a non-convex quadratic program. Since the optimization variable takes real values, we observed that in some cases more than one physical core was allocated to a virtual core–a solution that is not suitable for our one-to-one mapping formulation.

Then, we formulated the problem as a binary quadratic program, where the optimization variable is a vector which

express only mapping (as in [25]). This technique correctly handles mappings of a single physical core to a virtual core, avoiding the problem presented by the second approach.

3.1.2 Binary Quadratic Programming Problem Formulation

To reduce the scope of the problem, we formulated the problem as a sequence of non-convex binary quadratic programming problems, where each level of the problem mapped virtual cores to a level of system resources. In the first level, virtual machines (VMs) are mapped to sockets (SKs). In the second level virtual cores (VCs) belonging to those virtual machines are mapped to NUMA nodes (NMs). Finally, in a third level of mapping, virtual cores previously mapped to a NUMA node are mapped to physical cores (PCs). At each level, we estimate the performance impact of interference between co-located workloads and do not take in account potential cooperative behavior.

Our QP formulation is inspired by Sheng et al. [25]. We first defined constraints to ensure that a virtual core was mapped onto precisely one physical core, and that the maximum utilization of

$$\forall i \epsilon V \sum_{j=0}^{N_p} x_{ij} = 1 \quad (1)$$

$$\forall j \epsilon P \sum_{i=0}^{N_v} U_{ij} x_{ij} \leq 100 \quad (2)$$

where V is the set of virtual cores, P the set of physical cores, N_v the number of virtual cores, N_p the number of physical cores, U_{ij} indicates the percentage of physical core j allocated to virtual core i (this is a real value between 0 and 100), and x_{ij} is set to 1 when virtual core i is mapped into physical core j.

We then defined an objective function at each level that mapped the result of the allocation to projected system performance, based on expected VM and core interference if mapped to the same socket, NUMA domain, or core. For example, the objective function for mapping VMs to sockets used was:

$$min \sum_{u=0}^{Nvm} \sum_{v=0}^{Nvm} \sum_{s=0}^{Nsk} \sum_{t=0}^{Nsk} (I_{VMS}(u,v) S(s,t)) x_{us} x_{vt} \quad (3)$$

where N_{vm} is the number of virtual machines, N_{sk} the number of sockets, $I_{VMS}(u,v)$ indicates the interference between two virtual machines when allocated to the same socket, $S(s,t)$ is 1 if $s = t$ or 0 otherwise, and x_{ij} is 1 if VM i is mapped to socket j.

3.1.3 Performance Results

We solved this optimization problem using Matlab's binary quadratic programming solver described in [3]. For this experiment we used the Susitna-PRobE machines [10], which contain 4 x86 AMD Opteron(TM) 6272 processors, for a total of 64 cores and 8 NUMA nodes. These machines ran Ubuntu 12.04 LTS. We used the Mantevo suite's benchmarks [11] MiniMD, MiniFE, HPCCG, and CoMD and run 8 VMs each one with 4 VCs. A coefficient of sensitivity to interference was calculated for each memory domain (i.e. socket, NUMA, physical core) by creating contention with the sledge benchmark described in [20].

Our goal with these experiments is to determine how well an approximate solution to this optimization problem compares to a known-good solution in a simple case. If the approximation cannot handle simple cases, it is unlikely to perform well in more complex cases (e.g. with power, energy, and bandwidth bounds). On the other hand, if this approach is comparable to a hand-generated mapping in simple cases, it may be promising to examine its ability to handle more complex cases where what constitutes an "optimal" schedule is much less clear.

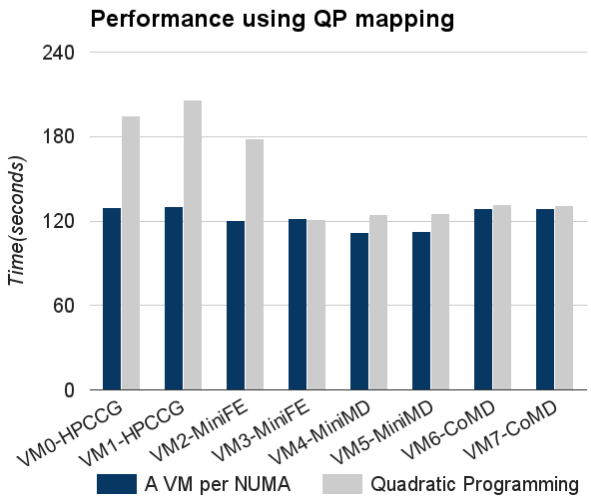

Figure 4: Mini-applications performance when allocated with the quadratic programming mapper.

Figure 4 shows the performance of the VMs mapped using the quadratic programming mapper compared with a mapping with no contention. Except for VMs 0 to 2, the performance of the mini-applications is very close to the ideal case in which they run in isolation within a NUMA domain. Performance degradation in the first three VMs is a result of the mapping solution obtained, which is a local minima solution, given to the non-convex nature of the problem.

Overall, these results demonstrate the potential viability of using numerical optimization techniques to allocate resources in HPC systems. An ideal mapping strategy is not a feasible approach due to its calculation cost, but our QP mapping approach achieves comparable performance while remaining computationally feasible. The results also demonstrate the challenges faced by these techniques. In particular, expressing scheduling problems in a way that fully encompasses the problem to be solved while remaining computationally feasible is challenging. New techniques emphasizing different formulations or different approximations that achieve solutions that are "good enough" are potentially of high impact.

3.2 Synchronization Mechanisms

Next, we implemented an EDF-based gang scheduler similar to that proposed by `vsched` [16] in order to examine its ability to schedule enclaves with synchronization/coordination demands. In this case, we mapped all virtual cores which run the enclaves of the gang to different physical cores.

We use partitioned EDF schedulers on each logical core, in order to offer gang scheduling capabilities. We set *slice (S)* and *period (T)* for all the virtual cores running applications of the gang to the same value.

Experiments were run on a Dell PowerEdge R15 machine with two AMD Opteron(tm) 4170 HE processors. This machine has two sockets, each socket has a single NUMA domain and 6 (2100 Mhz) cores. We used four physical cores for our experiments. For all the experiments, we launched two virtual machines, each one with four virtual cores. Each virtual core has a CPU utilization of approximately 38%. We mapped each virtual core to a separate physical core (cores 0 to 3) in NUMA domain 0. Thus, we mapped two virtual cores to each physical core and they did not migrate to other physical cores during the experiment. This setup resulted in 76% utilization for each physical core. Figure 5 shows the topology used for the experiments.

For each run, we launched an instance of the same benchmark on both virtual machines, in this case using the MiniMD and the MiniFE applications from Mantevo suite [11] and the FT, LU, SP, and BT of NAS benchmarks [21]. We configured the MiniMD application with a problem size of 4 4 4, 3000 iterations and 256 atoms, the miniFE application with 80*80*80 grid points, and we used class A versions of NAS (NPB3.3) benchmarks. Each benchmark instance ran four MPI processes in separate virtual cores.

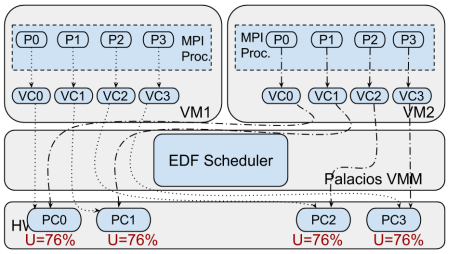

Figure 5: Topology for gang EDF scheduler experiments

As a baseline test, we ran the described benchmarks on the Palacios VMM [14] using its default scheduler. Then, we ran a second experiment using the EDF scheduler. In order to avoid synchronization, we configured different values of *slices* and *periods* for the virtual cores of each VM based on equation 4.

$$VC0 = \frac{S}{T} = VC1 = \frac{2S}{2T} = VC2 = \frac{3S}{3T} = VC3 = \frac{4S}{4T} \quad (4)$$

Notice that the *utilization factor* is the same for each physical processor. We will refer to this configuration as *Asynchronous EDF*. In a third experiment, we used the same values of *slice* and *period* for the virtual cores of the same virtual machine, in order to force them to run at the same time. We will refer to this configuration as *Gang EDF*.

For both EDF configurations, we set S to $S = 50ms$ and T to achieve the desired virtual core utilization.

Figure 6 shows the benchmark execution time normalized to the time of the default Palacios VMM scheduler. It shows the importance of a proper setting of the *slice* and *period* parameters: if these parameters are different for the virtual cores in the VM (Asynchronous EDF), there was a dramatic degradation in performance; if the *slice* and *period* parameters are the same for the virtual cores in the gang (Gang EDF), performance improved.

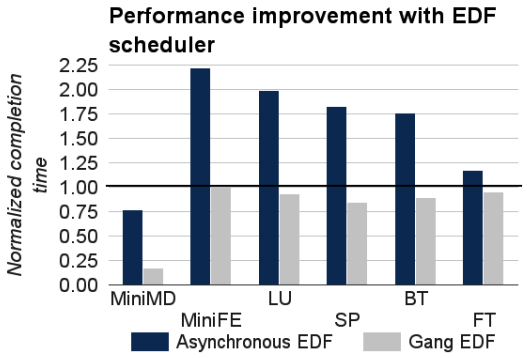

Figure 6: Performance improvement of gang EDF implementation over the asynchronous EDF scheduler

3.3 Co-location of Cooperative Enclaves

Finally, we conducted additional experiments to explore the impact of co-locating an additional work to a gang-scheduled core, simulating the effect of co-locating analytics without cooperatively scheduling the workload to minimize interference. To do this, we added a third VM to the gang scheduler experiment of Section 3.2; that VM runs the HPCCG benchmark in a single virtual core, using only 9% of the physical core. As a result, the utilization factor for that processor was 85% while the utilization factors of the other three processors remain 76%.

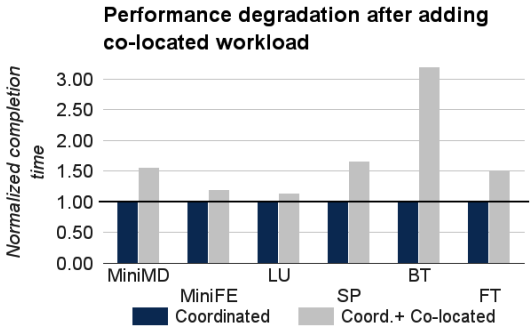

Figure 7: Performance degradation when a coordinated workload is co-located with an additional workload

The virtual cores that share physical core with the new added virtual core did not experience missed deadlines, which means that they still received their full CPU reservation. However, the addition of a new workload affected the synchronization of the gang-scheduled workloads. Figure 7 shows the degradation in performance for the studied benchmarks. The completion time is normalized to the completion time of the benchmarks without the addition of the new workload. Performance degraded for each application; the most sensitive application was the BT benchmark which experienced

a 200% increase in runtime.

4. RESEARCH CHALLENGES

The results in Section 3 illustrate a number of important research challenges that must be addressed for new exascale scheduling architectures. In particular, the combination of time-sharing of cores, virtualization, and cooperative scheduling presents difficult challenges on next-generation architectures. In the remaining portion of this section we discuss these scheduling challenges.

4.1 Workload Coordination/Synchronization Mechanisms

The results of the previous section demonstrate both the potential advantages of cooperative and coordinated scheduling, as well as key challenges in that area. In particular, coordinated scheduling (whether for coupled codes or co-located virtual machines) can make a dramatic performance difference for HPC applications. However, cooperative scheduling of these enclaves, where one enclave explicitly yields to another outside of the coordinated schedule, can completely negate any benefits of simple coordinated mechanisms. New mechanisms are needed that take into account both cooperative and coordinated scheduling of co-located HPC enclaves. Such mechanisms are difficult, however, because they must work cross-core or even potentially cross-nodes.

Figure 6 demonstrates the potential for a *vsched* [16] technique to introduce helpful synchronization mechanisms to coordinated workloads. However, co-located enclaves introduce additional complexity such as uneven utilization factors across cores. Further research to reduce overhead and improve scalability is needed. For example, the reduction of timer interrupts and its synchronization across cores as proposed in [12] has shown great promise. Moreover, low-overhead global coordination mechanisms should be investigated for inter-node aspects.

Figure 7 shows how the co-location of related workloads hurts performance when their cooperation is not considered by the scheduler. Additional mechanisms are needed in order to consider these relationships, as well as the dynamic nature on the performance requirements of these workloads. For example, the node-level scheduler may adapt scheduling parameters to provide tight synchronization when needed; however, these changing conditions may cause some level of overhead that should be studied.

4.2 Node-Level Resource Mapping Considerations

Scheduling policies that can schedule co-located, cooperative enclaves for next-generation HPC systems are needed. While a number of authors have proposed constrained optimization techniques for addressing some of these problems, additional research is needed to realize this approach, both in terms expressing realistic scheduling problems in these frameworks, and in solving the resulting problems. In addition, cooperative scheduling techniques also impact scheduling policy, as it influences the extent to which co-located enclaves interfere.

The results shown in Figure 4 show that the resource allocation problem formulated as a non-convex quadratic programming problem generates high quality though suboptimal solutions to simple scheduling problems. The viability of extending this formulation to more complex optimization criteria and resource constraints is, however, an open research question. Additional research is needed for more complex application interactions (e.g. related simulation and analytics workloads and gang scheduling), and express additional optimization constraints like node and system power caps.

4.3 Power Concerns

Power management presents a wide range of problems to HPC scheduling systems. Dynamic Voltage and Frequency Scaling (DVFS) techniques influence both coordination within a core as well as complicated scheduling policy decisions. For example, asymmetric DVFS decisions across cores can potentially have large negative performance impacts on coordinated scheduling. Similarly, scheduling policy on each core and across cores should also factor how DVFS impacts application performance. Because DVFS can impact the relative speed of the processor and memory systems, it could significantly impact any performance predictions used for driving scheduling policy.

5. RELATED WORK

5.1 High-Level Policies Support

Ramant et al. in [23] present a framework which uses *ClassAds* to define resource allocation policies on heterogeneous environments for workloads with co-location needs. The grid computing solution, Globus Toolkit [9] defines monitoring and discovery functions, which enable reporting information about physical resources through query or subscription mechanisms and use that information for discovery purposes. These mechanisms are supported in web-service based WSRF and WS-notification interfaces. Another language used to express requirements of jobs assigned to physical resources used in grid environments is the Job Submission Description Language (JSDL) [1]. These works could be used to define appropriate methods to express high-level policies and translate them to short-term mechanisms.

5.2 Node-level Resource Allocation

Rao et al. in [24] propose a NUMA-aware, contention-aware scheduler for virtualized systems. They use a metric based in the cost of the remote memory accesses, which is computed based on some hardware performance monitor (PMU) metrics. The drawback of this approach is that they must modify the guest kernel. Quadratic Programming approaches [4] propose a performance model which produces fast incremental, optimal solutions. A solution based on these approaches would permit the node-level resource allocation strategies needed to support loose provisioning requests.

5.3 Scheduling of Cooperative Workloads

Zheng et al. [30] attempt to run cooperative scientific workloads by time-sharing resources. This approach efficiently takes advantage of idle times of simulation workloads to run analysis workloads. It reduces contention, reduces data movement, and optimizes power consumption. Although this approach does not consider cooperative applications running on different operating systems or virtualized environments, the addressed problem is closely related to our work.

Chang et al. in [7] propose an approach in which workloads are classified as I/O-intensive or CPU-intensive. In high speed networks, where the costs of processing huge amount of packets are elevated, I/O-intensive tasks may not receive enough access to CPU resources, which are usually allocated to CPU-intensive tasks. They propose a solution in which CPU-intensive tasks voluntarily yield CPU cycles to I/O-intensive tasks. Similar mechanisms could be implemented in our approach in order to support scheduling of cooperative workloads.

5.4 Gang Scheduling

Kato et al. in [13] propose a gang EDF scheduler for multithread applications on multicore systems. In this approach, they claim that all the threads of a multithread application must be scheduled together. For that, they enhanced the global EDF policy by scheduling the set of threads of the same application at the same time, only when there are enough physical cores available. When there are not enough cores available, threads must wait for a time slice with sufficient core availability. This approach may cause CPU fragmentation, priority inversion [15] and execution delay. In order to solve these problems, Sukwong et al. in [26] propose a scheduler based built on top of KVM's Completely Fair Scheduler (CFS) in which sibling virtual cores are balanced to different physical cores.

The VMWare's relaxed co-scheduling [27] consists of a non-strict co-scheduling of virtual cores plus some mechanisms to reduce the skew of lagged virtual cores. It is implemented on the top of a proportional share algorithm.

We consider that synchronized per-core real time schedulers could improve these approaches. This enables real-time deadline guarantees, minimum QoS levels and deal with the co-location with cooperative applications through dynamic adjustment of tasks' reservation rates.

Bin Lin et al. in *vsched* work [16] present an approach based on per-node EDF schedulers which provide gang scheduling mechanisms by setting the same slice and period values to all the applications in the gang. This approach is optimal when space-shared resource allocation is used and even works well under some degree of contention. Our approach attempts to solve the problem of workloads with synchronization needs given time-shared CPUs with cooperative workloads. This is a more complex problem where we need to add further enhancements to EDF in order to meet optimization goals of the set of workloads.

In our earlier work [12] we present a run-time system based approach to avoid interference over large fine-grained parallel applications generated by short-lived system tasks. The run-time coordinates system tasks in order to run them at the same time, which helps to decrease interference in a meaningful way, allowing parallel applications to be co-scheduled at precise times, which improve their performance and scalability. This approach works for both inter-node and intra-node cases. A similar approach may be adopted to co-locate coordinated and cooperative enclaves. The use of a real time scheduler like an EDF scheduler may help to guarantee a more precise synchronization.

5.5 Real Time Scheduling

Bin Lin et al. in [17] propose periodic real time schedulers running on local nodes, coordinated by a global controller through feedback mechanisms. The global controller sets the target execution rate for each application, and adjusts it dynamically, based on the feedback received from the local schedulers. The technique attempts to adjust the application CPU utilization toward the target execution rate. This approach is not focused on HPC environments, where workload requirements have a dynamically changing behavior. Moreover, this model is inflexible in that it assumes nodes with the same target utilization rate. As an improvement, the provisioner could supply optimization policies and constraints and not specific execution rates; then the local node schedulers could translate those high-level policies to short-term scheduling mechanisms.

Zhang et al. in [29] present schedulability analyses of hierarchically scheduled multi-core systems where real time schedulers are used as local schedulers. Our approach uses this concept in high performance virtualized environments with complex performance goals and constraints.

6. CONCLUSION & FUTURE WORK

In this document we presented emerging scheduling challenges associated with large-scale high performance applications. We focused on the node-level mechanisms that new frameworks must provide to schedule selected HPC applications. Among the forthcoming needs are the ability to optimize performance or energy across complex exascale systems. We quantified the impact of the new scheduling challenges faced by extreme-scale systems. As future work, we plan to investigate a set of scheduling challenges which include: How do we efficiently integrate cooperative and coordinated scheduling across multiple nodes? Which coordination/synchronization mechanisms between node-level schedulers must be provided by the new frameworks? How do we efficiently handle high-level provisioning policies through low-level scheduling mechanisms? What kind of interfaces must be provided by the node-level schedulers to the provisioner? Which metrics must be reported by the node-level schedulers to the provisioner?

Acknowledgments

This work was supported in part by the 2013 Exascale Operating and Runtime Systems Program from the DOE Office of Science, Advanced Scientific Computing Research, under award number DE-SC0005050, program manager Sonia Sachs, and by the Colciencias-Fulbright Colombia and The Universidad Autonoma de Occidente through the Caldas scholarships program.

7. REFERENCES

[1] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva. Job Submission Description Language (JSDL) Specification, version 1.0. In *Open Grid Forum, GFD*, volume 56, 2005.

[2] P. Beckan, R. Brightwell, and A. B. Maccabe. Exascale operating systems and runtime software report. `http://science.energy.gov/~/media/ascr/pdf/research/cs/Exascale\%20Workshop/ExaOSR-Report-Final.pdf`, 2012.

[3] A. Bemporad. A Matlab function for solving Mixed Integer Quadratic Programs, version 1.02, user guide. `ftp://ftp.feq.ufu.br/Luis/HYBRID/MIP/`

hybtbx-win/doc/other/miqp.pdf, Last accesed 09/16/2012.

[4] S. L. Bird and B. J. Smith. PACORA: Performance aware convex optimization for resource allocation. In *Proceedings of the 3rd USENIX Workshop on Hot Topics in Parallelism (HotPar: Posters)*, 2011.

[5] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2009.

[6] R. Brightwell, R. Oldfield, A. B. Maccabe, and D. E. Bernholdt. Hobbes: Composition and virtualization as the foundations of an extreme-scale OS/R. In *Proceedings of the 3rd International Workshop on Runtime and Operating Systems for Supercomputers*, page 2. ACM, 2013.

[7] Z. Chang, J. Li, R. Ma, Z. Huang, and H. Guan. Adjustable credit scheduling for high performance network virtualization. In *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*, pages 337–345. IEEE, 2012.

[8] K. B. Ferreira, R. Brightwell, and P. G. Bridges. Characterizing application sensitivity to OS interference using kernel-level noise injection. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing (SC'08)*, November 2008.

[9] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of High Performance Computing Applications*, 11(2):115–128, 1997.

[10] G. Gibson, G. Grider, A. Jacobson, and W. Lloyd. Probe: A thousand-node experimental cluster for computer systems research. *USENIX*, 38(3), 2013.

[11] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich. Improving performance via mini-applications. *Sandia National Laboratories, Tech. Rep*, 2009.

[12] T. Jones, S. Dawson, R. Neely, W. Tuel, L. Brenner, J. Fier, R. Blackmore, P. Caffrey, B. Maskell, P. Tomlinson, et al. Improving the scalability of parallel jobs by adding parallel awareness to the operating system. In *Supercomputing, 2003 ACM/IEEE Conference*, pages 10–10. IEEE, 2003.

[13] S. Kato and Y. Ishikawa. Gang EDF scheduling of parallel task systems. In *Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE*, pages 459–468. IEEE, 2009.

[14] J. Lange, K. Pedretti, T. Hudson, P. Dinda, Z. Cui, L. Xia, P. Bridges, A. Gocke, S. Jaconette, M. Levenhagen, et al. Palacios and kitten: new high performance operating systems for scalable virtualized and native supercomputing. In *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–12. IEEE, 2010.

[15] W. Lee, M. Frank, V. Lee, K. Mackenzie, and L. Rudolph. Implications of I/O for Gang scheduled workloads. In *Job Scheduling Strategies for Parallel Processing*, pages 215–237. Springer, 1997.

[16] B. Lin and P. A. Dinda. Vsched: Mixing batch and interactive virtual machines using periodic real-time scheduling. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 8. IEEE Computer Society, 2005.

[17] B. Lin, A. I. Sundararaj, and P. A. Dinda. Time-sharing parallel applications with performance isolation and control. In *Autonomic Computing, 2007. ICAC'07. Fourth International Conference on*, pages 28–28. IEEE, 2007.

[18] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[19] P. Luszczek, J. J. Dongarra, D. Koester, R. Rabenseifner, B. Lucas, J. Kepner, J. McCalpin, D. Bailey, and D. Takahashi. Introduction to the HPC challenge benchmark suite. *Lawrence Berkeley National Laboratory*, 2005.

[20] J. Mars, L. Tang, and M. L. Soffa. Directly characterizing cross core interference through contention synthesis. In *Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers*, pages 167–176. ACM, 2011.

[21] NAS. NAS parallel benchmarks. `https://www.nas.nasa.gov/publications/npb.html`, Last accessed, February 2014.

[22] F. A. Omara and M. M. Arafa. Genetic algorithms for task scheduling problem. *Journal of Parallel and Distributed Computing*, 70(1):13–22, 2010.

[23] R. Raman, M. Livny, and M. Solomon. Policy driven heterogeneous resource co-allocation with gangmatching. In *High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on*, pages 80–89. IEEE, 2003.

[24] J. Rao, K. Wang, X. Zhou, and C.-Z. Xu. Optimizing virtual machine scheduling in NUMA multicore systems. In *HPCA*, pages 306–317, 2013.

[25] J. Sheng, L. Zhong, Z. Yu, X. Zeng, et al. A method of quadratic programming for mapping on NoC architecture. In *ASIC (ASICON), 2011 IEEE 9th International Conference on*, pages 200–203, 2011.

[26] O. Sukwong and H. S. Kim. Is co-scheduling too expensive for SMP VMs? In *Proceedings of the sixth conference on Computer systems*, pages 257–272. ACM, 2011.

[27] VMWare. The CPU Scheduler in VMware vSphere 5.1. Technical White Paper. `https://www.vmware.com/files/pdf/techpaper/VMware-vSphere-CPU-Sched-Perf.pdf`, Last accessed, 02/2014.

[28] O. U. P. Zapata and P. M. Alvarez. EDF and RM multiprocessor scheduling algorithms: Survey and performance evaluation. *Seccion de Computacion Av. IPN*, 2508, 2005.

[29] F. Zhang and A. Burns. Analysis of hierarchical EDF pre-emptive scheduling. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 423–434. IEEE, 2007.

[30] F. Zheng, H. Yu, C. Hantas, M. Wolf, G. Eisenhauer, K. Schwan, H. Abbasi, and S. Klasky. Goldrush: resource efficient in situ scientific data analytics using fine-grained interference aware execution. In *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, page 78. ACM, 2013.